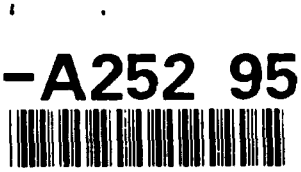

(2)

AD

TECHNICAL REPORT ARCCB-TR-92021

ADAPTIVE METHODS AND PARALLEL COMPUTATION FOR PARTIAL DIFFERENTIAL EQUATIONS

RUPAK BISWAS
MESSAOUD BENANTAR
JOSEPH E. FLAHERTY

MAY 1992

US ARMY ARMAMENT RESEARCH,
DEVELOPMENT AND ENGINEERING CENTER
CLOSE COMBAT ARMAMENTS CENTER
BENÉT LABORATORIES
WATERVLIET, N.Y. 12189-4050

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The use of trade name(s) and/or manufacturer(s) does not constitute an official indorsement or approval.

DESTRUCTION NOTICE

For classified documents, follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX.

For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

For unclassified, unlimited documents, destroy when the report is no longer needed. Do not return it to the originator.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1992	Final

4. TITLE AND SUBTITLE
ADAPTIVE METHODS AND PARALLEL COMPUTATION FOR PARTIAL DIFFERENTIAL EQUATIONS

5. FUNDING NUMBERS
AMCMS: 6111.02.H610.011
PRON: 1A04Z0CANMSC

6. AUTHOR(S)
Rupak Biswas (RPI, Troy, NY), Messaoud Benantar (RPI), and Joseph E. Flaherty (RPI and Benet)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army ARDEC
Benet Laboratories, SMCAR-CCB-TL
Watervliet, NY 12189-4050

8. PERFORMING ORGANIZATION REPORT NUMBER
ARCCB-TR-92021

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army ARDEC
Close Combat Armaments Center
Picatinny Arsenal, NJ 07806-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Presented at the Eighth Army Conference on Applied Mathematics and Computing, Cornell University, Ithaca, NY, 19-22 June 1990.
Published in Proceedings of the Conference.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Consider the adaptive solution of two-dimensional vector systems of hyperbolic and elliptic partial differential equations on shared-memory parallel computers. Hyperbolic systems are approximated by an explicit finite volume technique and solved by a recursive local mesh refinement procedure on a tree-structured grid. Local refinement of the time steps and spatial cells of a coarse base mesh is performed in regions where a refinement indicator exceeds a prescribed tolerance. Computational procedures that sequentially traverse the tree while processing solutions on each grid in parallel, that process solutions at the same tree level in parallel, and that dynamically assign processors to nodes of the tree have been developed and applied to an example. Computational results comparing a variety of heuristic processor load balancing techniques and refinement strategies are presented.

14. SUBJECT TERMS
Adaptive Systems, Partial Differential Equations, Local Mesh Refinement, Meshes, Refinement, Load Balancing Techniques, Finite Element Analysis

15. NUMBER OF PAGES
22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ADAPTIVE METHODS AND PARALLEL COMPUTATION FOR PARTIAL DIFFERENTIAL EQUATIONS*

Rupak Biswas, Messaoud Benantar

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

and

Joseph E. Flaherty

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
and
U.S. Army Armament, Munitions, and Chemical Command
Armament Research, Development, and Engineering Center
Close Combat Armaments Center
Benét Laboratories
Watervliet, NY 12189

ABSTRACT. Consider the adaptive solution of two-dimensional vector systems of hyperbolic and elliptic partial differential equations on shared-memory parallel computers. Hyperbolic systems are approximated by an explicit finite volume technique and solved by a recursive local mesh refinement procedure on a tree-structured grid. Local refinement of the time steps and spatial cells of a coarse base mesh is performed in regions where a refinement indicator exceeds a prescribed tolerance. Computational procedures that sequentially traverse the tree while processing solutions on each grid in parallel, that process solutions at the same tree level in parallel, and that dynamically assign processors to nodes of the tree have been developed and applied to an example. Computational results comparing a variety of heuristic processor load balancing techniques and refinement strategies are presented.

For elliptic problems, the spatial domain is discretized using a finite quadtree mesh generation procedure and the differential system is discretized by a finite element-Galerkin technique with a hierarchical piecewise polynomial basis. Adaptive mesh refinement and order enrichment strategies are based on control of estimates of local and global discretization errors. Resulting linear algebraic systems are solved by a conjugate gradient technique with a symmetric successive over-relaxation

* This research was partially supported by the U. S. Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant Number AFOSR-90-0194; by the SDIO/IST under management of the U. S. Army Research Office under Contract Number DAAL03-90-G-0096; and by the National Science Foundation under Grant Numbers CDA-8805910 and CCR-8920694.

preconditioner. Stiffness matrix assembly and linear system solution are processed in parallel with computations scheduled on noncontiguous quadrants of the tree in order to minimize process synchronization. Determining noncontiguous regions by coloring the regular finite quadtree structure is far simpler than coloring elements of the unstructured mesh that the finite quadtree procedure generates. We describe a linear-time complexity coloring procedure that uses a maximum of six colors.

1. INTRODUCTION. Partial differential equations that arise in scientific and engineering applications typically feature solutions that develop, evolve, and decay on diverse temporal and spatial scales. The Fokker-Planck equation of mathematical physics may be used to illustrate this phenomenon. Perspective renditions of its solution u as a function of two spatial arguments x and y are shown at four times t in Figure 1 [20]. As time progresses, a single "spike" in the probability density arising from an initial Maxwell-Boltzmann distribution evolves into the two spikes shown. Conventional fixed-step and fixed-order finite difference and finite element techniques for solving such problems would either require excessive computing resources or fail to adequately resolve nonuniform behavior. As a result, they are gradually being replaced by adaptive methods that offer greater efficiency, reliability, and robustness by automatically refining, coarsening, or relocating meshes or by varying the order of numerical accuracy.

Adaptive software for ordinary differential equations has existed for some time and procedures that vary both mesh spacing and order of accuracy are in common use for both initial [17] and boundary [7] value problems. The situation is far more difficult for partial differential equations due to the greater diversity of phenomena that can occur; however, some production-ready adaptive software has appeared for elliptic problems [12]. The state of the art for transient problems lags that for elliptic systems but some projects are underway [16]. Adaptive strategies will either have to be retrofitted into an existing software system for solving partial differential equations or have to be coupled with pre- and post-processing software tools before widespread use occurs.

With an adaptive procedure, an initial crude approximate solution generated on a coarse mesh with a low-order numerical method is enriched until a prescribed accuracy level is attained. Adaptive strategies in current practice are classified as h-, p-, or r-refinement when, respectively, computational meshes are refined or coarsened in regions of the problem domain that require more or less resolution [6, 12], the order of accuracy is varied in different regions [10], or a fixed-topology mesh is redistributed [5]. These basic enrichment methods may be used alone or in combination. The particular combination of h- and p-refinement, for example, has been shown to yield exponential convergence rates in certain situations [9].

Enrichment indicators, which are frequently estimates of the local discretization error of the numerical scheme, are used to control the adaptive process. Resources (finer meshes, higher-order methods, etc.) are introduced in regions having large enrichment indicators and deleted from regions where indicators are low. Using estimates of the discretization error as enrichment indicators also enables the calculation

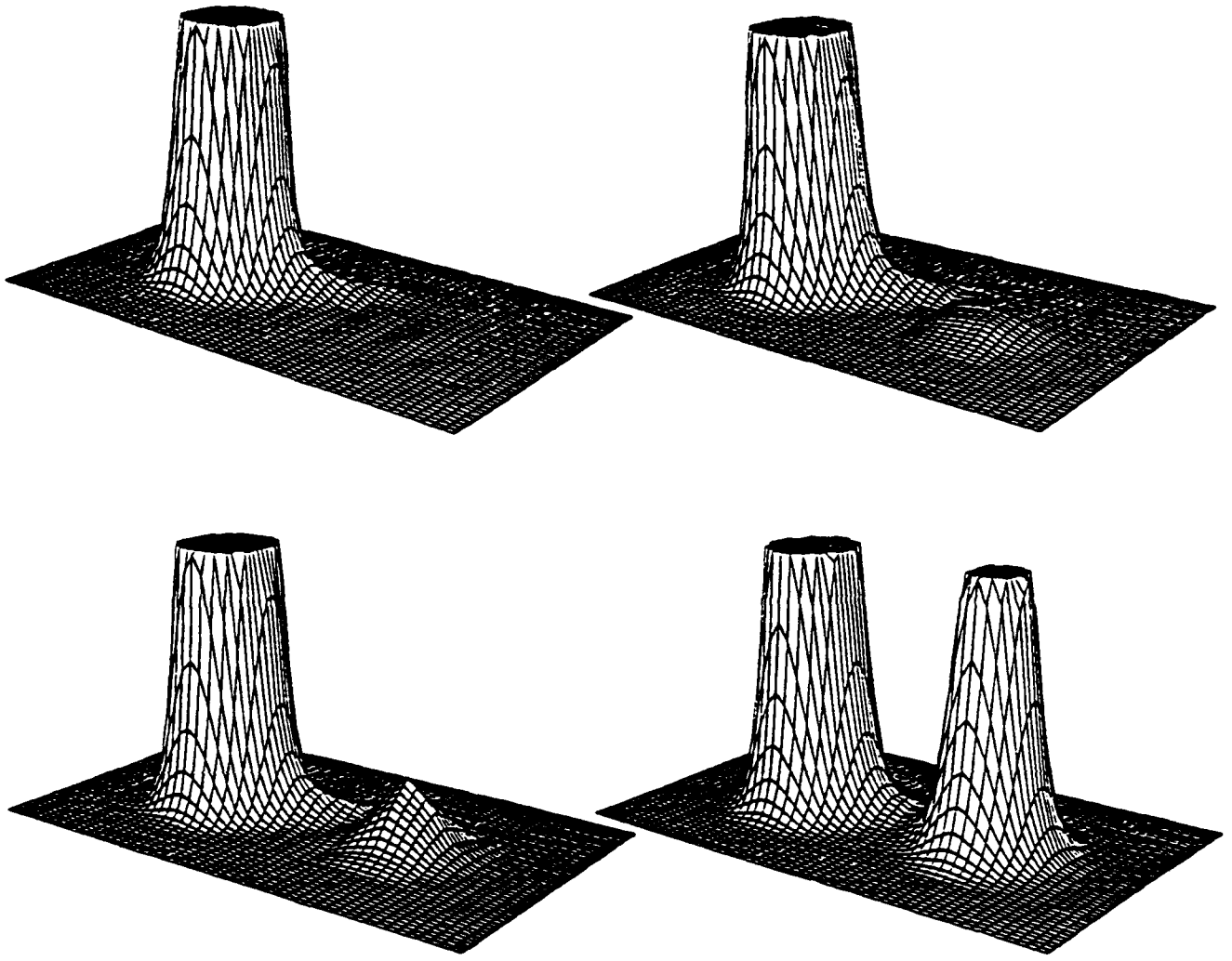

Figure 1. Solution $u(x,y,t)$ of the Fokker-Planck equation at times $t = 4$ (upper left), 10 (upper right), 20 (lower left), and 100 (lower right) obtained by Moore and Flaherty [20]. Solutions having magnitudes greater than 0.1 have been omitted in order to emphasize fine-scale structure.

of local and global accuracy measures which should become a standard part of every scientific computation. Estimates of the local discretization error are typically obtained by using either h- or p-refinement. Thus, one uses the difference between solutions computed either on two meshes or with two distinct orders of accuracy to furnish an error estimate. Special "superconvergence" points where solutions converge faster than they do globally can be used to significantly reduce computational cost [2].

Parallel procedures are becoming increasingly important both as hardware systems become available and as problem complexity increases. Furthermore, the efficiency afforded by adaptive strategies, cannot be ignored in a parallel computational

environment since the demand to model nature more accurately is always beyond hardware capabilities. Models of parallel computation are based on distributed-memory and shared-memory architectures. Distributed-memory systems tend to have large numbers of relatively simple processing elements connected in a network. Available memory on these fine-grained systems is distributed with the processing elements at the nodes of the network, so data access is by message passing. Balancing communication and synchronizing processing is extremely important because processing elements are typically operating in lock-step fashion in order to improve throughput and processor utilization. Shared-memory systems involve a more coarse-grained level of parallelism with relatively few processors operating asynchronously and communicating with a global memory, although variations are common. For example, processing elements may have a local cache memory in order to reduce bus contention and may have vector capabilities; thus, providing a hierarchy of coarse- and fine-grained parallelism.

Our goal is to develop parallel adaptive methods for partial differential equations. Fortunately, our adaptive software utilizes hierarchical (tree) data structures that have many embedded parallel constructs. Transient hyperbolic problems may generally be solved using explicit numerical techniques which greatly simplify processor communication. Experiments, reported in Section 2, with a variety of tree traversal strategies on an adaptive mesh refinement finite difference scheme [6] indicate that the dynamic load balancing scheme of assigning grid-vertex computations at a given tree level to processors as they become available provided the best parallel performance. Static load balancing strategies, that either traverse the tree of grids serially while processing solutions on each grid in parallel or traverse the tree in parallel while processing solutions on grids at the same tree level are also discussed. These alternatives to dynamic processor assignment may provide better performance on hierarchical memory computers.

For elliptic problems, system assembly and solution are processed in parallel with computations scheduled on noncontiguous tree quadrants in order to minimize process synchronization. "Coloring" the elements of a mesh so as to avoid memory contention on a shared-memory computer is far simpler when an underlying tree structure is present than for more general unstructured grids that the finite quadtree structure generates. The six-color procedure, described in Section 3, for the finite element solution scheme on quadtree-structured grids displays a high degree of parallelism when piecewise linear approximations are used. Unfortunately, the same procedure does not do as well with higher-order piecewise polynomial approximations; however, an element edge coloring procedure may improve performance.

2. HYPERBOLIC SYSTEMS. Consider a system of two-dimensional conservation laws in m variables on a rectangular domain Ω having the form

$$\mathbf{u}_t + \mathbf{f}_x(x,y,t,\mathbf{u}) + \mathbf{g}_y(x,y,t,\mathbf{u}) = 0, \quad (x,y) \in \Omega, \quad t > 0, \tag{1a}$$

subject to the initial conditions

$$\mathbf{u}(x,y,0) = \mathbf{u}^0(x,y), \quad (x,y) \in \Omega \bigcup \partial\Omega, \tag{1b}$$

and appropriate well-posed boundary conditions. The functions $\mathbf{u}$, $\mathbf{f}$, $\mathbf{g}$, and $\mathbf{u}^0$ are m-vectors, x and y are spatial coordinates, t denotes time, and $\partial\Omega$ is the boundary of Ω.

Our research is based on a serial adaptive hr-refinement algorithm of Arney and Flaherty [6]. We forego mesh motion at present and briefly describe an h-refinement procedure that utilizes their strategy. The problem (1) is solved on a coarse rectangular "base" mesh for a sequence of base-mesh time slices of duration Δt_n, $n = 0, 1, \cdots$, by an explicit finite difference, finite volume, or finite element scheme. For a base-mesh time step, say from t_n to $t_{n+1} = t_n + \Delta t_n$, a discrete solution is generated on the base mesh along with a set of local enrichment indicators which, in this case, are refinement indicators. Cells of the mesh where refinement indicators at t_{n+1} fail to satisfy a prescribed tolerance are identified and grouped into rectangular clusters. After ensuring that clusters have an adequate percentage of high-refinement-indicator cells and subsequently enlarging the clusters by a one-element buffer to provide a transition between regions of high and low refinement indicators, cells of the base mesh are bisected in space and time to create finer meshes that are associated with each rectangular cluster. Local problems are solved on the finer meshes and the refinement procedure is repeated until refinement indicators satisfy the prescribed unit-step criteria. After finding an acceptable solution on the base mesh, the integration continues with, possibly, a new base-mesh time step Δt_{n+1}. Data management involves the use of a tree structure with nodes of the tree corresponding to meshes at each refinement level for the current base-mesh time step. The base mesh is the root of the tree and finer grids are regarded as offspring of coarser ones.

With an aim of maintaining generality at the possible expense of accuracy and performance, we discretize (1) using the Richtmyer two-step version of the Lax-Wendroff method [23], which we describe for a one-dimensional problem as follows. Introduce a mesh on Ω having spacing $\Delta x_j = x_{j+1} - x_j$ and let the discrete approximation of $\mathbf{u}(x_j, t_n)$ be denoted as $\mathbf{U}_j^n$. Predicted solutions at cell centers are generated by the Lax-Friedrichs scheme, i.e.,

$$\mathbf{U}_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \tfrac{1}{2}(\mathbf{U}_{j+1}^n + \mathbf{U}_j^n) - \frac{\Delta t_n}{2\Delta x_j}(\mathbf{f}_{j+1}^n - \mathbf{f}_j^n). \tag{2a}$$

This provisional solution is then corrected by the leapfrog scheme

$$\mathbf{U}_j^{n+1} = \mathbf{U}_j^n - \frac{2\Delta t_n}{\Delta x_j + \Delta x_{j-1}}(\mathbf{f}_{j+\frac{1}{2}}^{n+\frac{1}{2}} - \mathbf{f}_{j-\frac{1}{2}}^{n+\frac{1}{2}}). \tag{2b}$$

Following Arney et al. [4, 6], refinement indicators are selected as estimates of the local discretization error obtained by Richardson's extrapolation (h-refinement) on a mesh having half the spatial and temporal spacing of the mesh used to generate the solution. Fine-mesh solutions generated as part of this error estimation process may subsequently be used on finer meshes when refinement is necessary. Initial and boundary data for refined meshes is determined by piecewise bilinear polynomial interpolation from acceptable solutions on the finest available meshes.

Parallel procedures are developed for the adaptive h-refinement solution scheme described above using a P-processor concurrent read exclusive write (CREW) shared-memory MIMD computer. We consider both static and dynamic strategies for balancing processor loading. As the names imply, with static load balancing, processors are assigned tasks a priori with the goal of having them all terminate at approximately the same time, whereas, with dynamic load balancing, available processors are assigned tasks from a task queue. Two possible static load balancing techniques come to mind: (i) serial depth-first traversal of the tree of grids with solutions on each grid being generated in parallel and (ii) parallel generation of solutions on all grids that are at the same tree level. With the depth-first traversal procedure, each grid is statically divided into P subregions and a processor is assigned to each subregion. With the parallel tree traversal procedure, the P processors are distributed among all grids at a particular tree level so as to balance loading. Thus, parallelism occurs both within a grid and across the breadth of the tree with this strategy. In both cases, the parallel solution process proceeds from one base-mesh time step to the next.

Serial depth-first traversal of the tree leads to a highly structured algorithm that has a straight-forward design because the same procedure is used on all grids. Balancing processor loading on rectangular grids is nearly perfect with an explicit finite difference scheme like (2) and similar behavior could be expected for geometrically complex regions. Load imbalance occurs due to differences in the time required to compute initial data. Other than at $t = 0$, initial data is determined by interpolating solutions from the finest grid at the end of the previous base-mesh time step to the present grid. Tree traversal, required to determine the correct solution vertices for the interpolation, would generally take different times in different regions due to variations in tree depth. This defect might be remedied by using either a more sophisticated domain decomposition technique or a more complex data structure to store the tree of grids.

The serial depth-first traversal procedure becomes inefficient when P is of the order of the number of elements in a grid. This situation can be avoided by refining grids by more than a binary factor; thus, maintaining a shallow tree depth. Lowering the efficiency of clusters by including a greater percentage of low-refinement-indicator cells will also increase grid size but diminish optimal grid usage. The inefficiency cited here should not be a factor on data-parallel computers and the serial tree traversal procedure might also be viable there.

The parallel tree-traversal procedure requires complex dynamic scheduling to assign processors to grids. One possibility is to estimate the work remaining to reduce error estimates to prescribed tolerances and to assign processors to subgrids so as to balance this load. Were such a heuristic technique successful, the parallel tree traversal procedure would not degrade in efficiency when the number of elements on a grid is $O(P)$.

Consider a situation where Q processors are used to obtain a solution on a grid at tree level $l - 1$ and suppose that refinement indicators dictate the creation of L grids $G_{l,i}$, $i = 1, 2, \cdots, L$, at level l. Further assume that

i. the prescribed local refinement tolerance at level $l-1$ is τ_{l-1};

ii. the areas of $G_{l,i}$ are $M_{l,i}$, $i = 1, 2, \cdots, L$;

iii. estimates $E_{l,i}$ of the discretization error are available for $G_{l,i}$, $i = 1, 2, \cdots, L$; and

iv. the convergence rate of the numerical scheme is known as a function of the local mesh spacing.

The Richtmyer two-step scheme (2) has a quadratic convergence rate which we use to illustrate the load balancing technique; however, the approach easily extends to other convergence rates.

In order to satisfy the prescribed accuracy criterion, $G_{l,i}$ should be refined by a factor of $(E_{l,i}/\tau_{l-1})^2$. The time step on $G_{l,i}$ must be reduced by a factor of $E_{l,i}/\tau_{l-1}$ in order to satisfy the Courant condition. Hence, the expected work $W_{l,i}$ to find an acceptable solution on $G_{l,i}$ is

$$W_{l,i} = M_{l,i}\left[\frac{E_{l,i}}{\tau_{l-1}}\right]^3. \tag{3}$$

The Q available processors should be allocated so as to balance the time required to complete the expected work on each of the L grids at level l. Thus, assign Q_i processors to grid $G_{l,i}$, $i = 1, 2, \cdots, L$, so that

$$\frac{W_{l,1}}{Q_1} = \frac{W_{l,2}}{Q_2} = \cdots = \frac{W_{l,L}}{Q_L}, \quad \sum_{i=1}^{L} Q_i = Q. \tag{4a,b}$$

The quality of load balancing using this approach will depend on the accuracy of the discretization error estimate. Previous investigations [4, 6] revealed that error estimates were generally better than 80 percent of the actual error for a wide range of mesh spacings and problems. Equation (3) can be used to select refinement factors other than binary and, indeed, to select different refinement levels for different meshes at a given tree level. This consideration combined with over-refinement to a tolerance somewhat less than the prescribed tolerance should maintain a shallow tree depth and enhance parallelism at the expense of grid optimality.

Simple dynamic load balancing can take full advantage of the CREW shared-memory MIMD environment. One just maintains a queue of mesh points at a given tree level and computes solutions at these points as processors become available. Load balancing is perfect except for any inherent system hardware anomalies. Balancing processor loads on geometrically complex regions is as simple as on rectangular regions because mesh points are processed on a first-come-first-serve basis independently of the grid to which they belong. Nonuniformities in initial data also introduce no problems and neither does the relationship of P to the number of cells in a grid. Finally, complex processor scheduling based on accurate error estimates is avoided. This strategy, however, might not be appropriate for hierarchical or distributed-memory computers.

Binary refinement of space-time grids may be optimal in using the fewest mesh points; however, tree depth tends to be large and this introduces serial overhead into a parallel procedure. As previously suggested, serial overhead can be reduced by keeping tree depth shallow and to do this we perform M-ary instead of binary refinement. The value of M is chosen adaptively for different clusters so that the prescribed tolerance is satisfied after a single refinement step. Thus, if τ_0 is the prescribed local discretization error tolerance, then choose M for grid $G_{1,i}$ as the first even integer greater than $E_{1,i}/\tau_0$. Having a good a priori knowledge of the work required on each cluster, processors can be distributed among the grids according to (4) to effectively balance loading. Of course, the refinement tolerance may not be satisfied after performing one level of M-ary refinement. Should this occur, we perform additional levels of 2-ary refinement until accuracy requirements are satisfied. The terms "binary" and "2-ary" refinement have been used to distinguish differences in our methods of checking the refinement condition. With binary refinement, the refinement condition is checked after each of the two finer time steps but with 2-ary refinement, the condition is only checked after the second time step. As a result, the fine grids remain unchanged for both of the two finer time steps with 2-ary refinement.

The efficiency of this mesh refinement strategy and of the serial depth-first traversal and dynamic balancing techniques are appraised in an example. Performance of the parallel traversal procedure was not as good as either of these schemes and results are not presented for it. Computer codes based on these algorithms have been implemented on a 16-processor Sequent Balance 21000 shared-memory parallel computer. Parallelism on this system is supported through the use of a parallel programming library that permits the creation of parallel processes and enforces synchronization and communication using barriers and hardware locks. CPU time and parallel speed up are used as performance measures.

Example 1. Consider the linear scalar differential equation

$$u_t + 2u_x + 2u_y = 0, \quad 0 < x, y < 1, \quad t > 0, \tag{5a}$$

with initial and Dirichlet boundary data specified so that the exact solution is

$$u(x,y,t) = \tfrac{1}{2}[1 - \tanh(100x - 10y - 180t + 10)]. \tag{5b}$$

The solution (5b) is a relatively steep but smooth wave that moves at an angle of 45 degrees across the square domain as time progresses.

Adaptive refinement is controlled by using an approximation of the local discretization error in the L^1 norm as a refinement indicator. Exact errors for this scalar problem are also measured in L^1 as

$$\|e(\cdot,\cdot,t)\|_1 = \iint_{\Omega} |\mathbf{P}u(x,y,t) - U(x,y,t)|\, dxdy, \tag{6}$$

where $U(x,y,t)$ is a piecewise constant representation of the discrete solution and $\mathbf{P}u(x,y,t)$ is a projection onto the space of piecewise constant functions obtained by using values at cell centers.

Our first experiment involves the solution of (5) for $0 < t \leq 0.35$ on 10×10, 25×25, and 45×45 uniform grids having initial time steps of 0.017, 0.007, and 0.004, respectively. No spatial refinement was performed and the static and dynamic load balancing strategies were used. CPU times and parallel speed ups for each base mesh for the two load balancing techniques are shown in Figure 2. Speed up with 15 processors and the static load balancing technique (shown in the upper portion of Figure 2) are in excess of 51, 75, and 87 percent of ideal with the 10×10, 25×25, and 45×45 base meshes, respectively. Speed up increases dramatically as the mesh is made finer due to smaller data granularity. Similar speed up data for the three base meshes with the dynamic load balancing technique (shown in the lower portion of Figure 2) are 53, 77, and 90 percent of ideal. The static load balancing strategy takes slightly more time than the dynamic technique, except in the uniprocessor case where they are identical, because of load imbalances on the P subdomains due to differences in the times required to generate initial and boundary data.

Our second experiment involves solving (5) for $0 < t \leq 0.35$ on a 10×10 base mesh having an initial time step of 0.017 using dynamic load balancing and adaptive h-refinement with either binary refinement or M-ary followed by 2-ary refinement. Refinement tolerances of 0.012, 0.006, and 0.003 were selected. The resulting CPU times and parallel speed ups for each adaptive strategy are presented in Figure 3. Maximum speed ups shown in the upper portion of Figure 3 for the binary refinement strategy are in excess of 82, 86, and 72 percent of ideal for tolerances of 0.012, 0.006, and 0.003, respectively. Initially, parallel performance improves as the tolerance is decreased due to the finer data granularity; however, the performance ultimately degrades due to the serial overhead incurred when managing a more complex data structure. Maximum speed ups for the more sophisticated M-ary followed by 2-ary refinement strategy shown in the lower portion of Figure 3 are in excess of 88, 82, and 73 percent of ideal for the three decreasing tolerances. Speed ups for this refinement strategy are only marginally better than those for the binary refinement technique, but the CPU times for the M-ary strategy are much less than those for the binary refinement strategy. For example, CPU times with 15 processors and a tolerance of 0.003 were 226.11 and 182.73 for the binary and M-ary strategies, respectively. Maintaining a shallow tree has clearly increased performance by reducing the serial overhead associated with its management.

Speed up is not an appropriate measure of the complexity required to solve a problem to a prescribed level of accuracy. Tradeoffs occur between the higher degree of parallelism possible with a uniform mesh solution and the greater sequential efficiency of an adaptive procedure. In order to gage the differential, we generated uniform mesh and adaptive mesh solutions of (5) on various processor configurations and to varying levels of accuracy for both static serial tree traversal and dynamic load balancing strategies. Computations on uniform grids ranged from a 5×5 mesh to a 45×45 mesh. All adaptive computations used a 10×10 base mesh, M-ary followed by 2-ary refinement, and tolerances ranging from 0.012 to 0.003.

Results for the global L^1 error as a function of CPU time are presented in Figure 4 for computations performed on 1, 4, 8, and 15 processor systems. Static and dynamic load balancing strategies are shown in the upper and lower portions of the

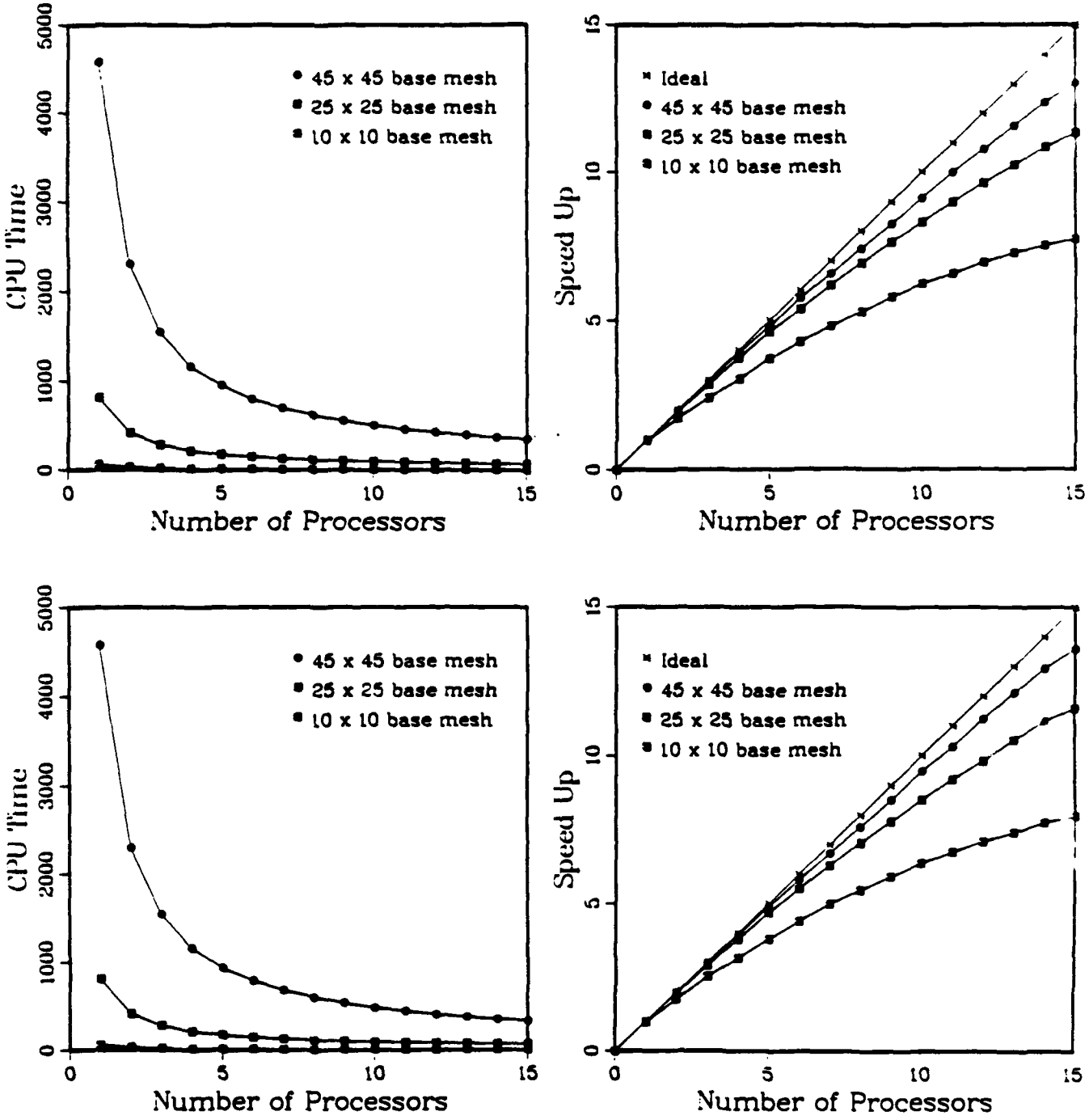

Figure 2. CPU time (left) and parallel speed up (right) for Example 1 on uniform meshes without adaptivity using static (top) and dynamic (bottom) load balancing.

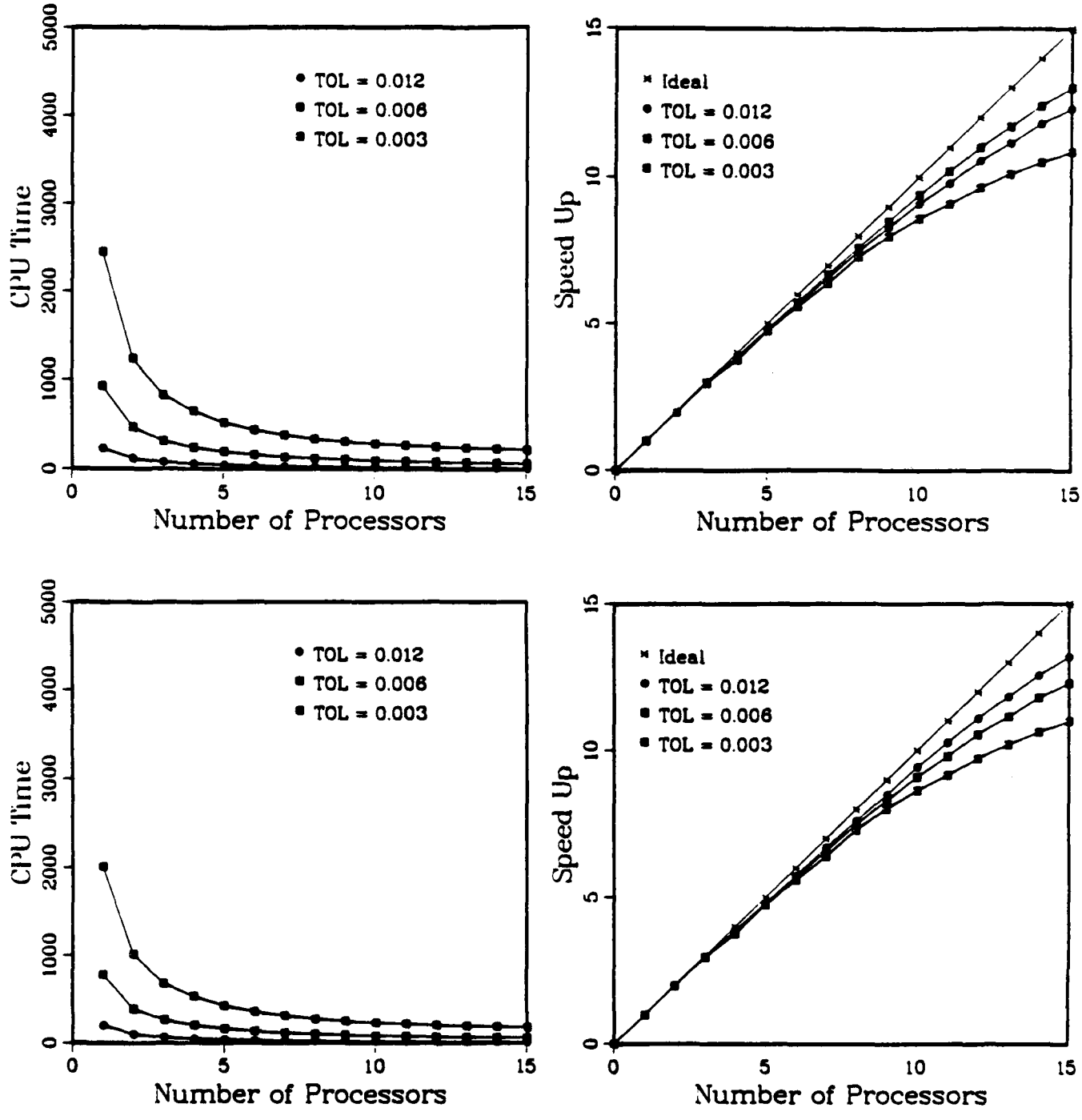

Figure 3. CPU time (left) and parallel speed up (right) for Example 1 using dynamic load balancing and adaptive h-refinement with local binary refinement (top) and M-ary followed by 2-ary refinement (bottom).

figure, respectively. For each strategy, the upper set of curves, displaying non-adaptive results, are much less efficient and converging at a much slower rate than the adaptive solutions shown in the lower set of curves. The adaptive solutions are converging at a rate of approximately 1.4 relative to CPU time while the non-adaptive solutions are converging at a rate of approximately 0.4. These results demonstrate a strong preference for adaptive methods for all but the largest tolerances. Note that the CPU times are identical for the two load balancing strategies when only one processor is used for both non-adaptive and adaptive solutions because the configuration reduces to that of a uniprocessor system. Also note that the global L^1 error for a particular choice of base mesh (for non-adaptive methods) or local refinement tolerance (for adaptive methods) is independent of the number of processors used.

3. ELLIPTIC SYSTEMS. With the goal of describing a strategy for solving linear algebraic systems resulting from the finite element discretization of elliptic systems, let us consider a two-dimensional linear elliptic problem in m variables having the form

$$-[\mathbf{D}(x,y)\mathbf{u}_x]_x - [\mathbf{D}(x,y)\mathbf{u}_y]_y + \mathbf{Q}(x,y)\mathbf{u} = \mathbf{f}(x,y), \quad (x,y) \in \Omega, \tag{7a}$$

$$u_i = c_i^E(x,y), \quad (x,y) \in \partial\Omega_i^E, \quad (\mathbf{D}\mathbf{u}_\nu)_i = c_i^N(x,y), \quad (x,y) \in \partial\Omega_i^N,$$
$$i = 1, 2, \cdots, m. \tag{7b,c}$$

The diffusion $\mathbf{D}$ and source $\mathbf{Q}$ are positive definite and positive semi-definite $m \times m$ matrices, respectively, $\partial\Omega = \partial\Omega_i^E \bigcup \partial\Omega_i^N$, $i = 1, 2, \cdots, m$, and ν is a unit outer normal to $\partial\Omega$.

The Galerkin form of (7) consists of determining $\mathbf{u} \in H_E^1$ satisfying

$$A(\mathbf{v},\mathbf{u}) + (\mathbf{v},\mathbf{f}) = \sum_{i=1}^{m} \int_{\partial\Omega_i^N} v_i c_i^N \, ds, \quad \text{for } \textit{all } \mathbf{v} \in H_0^1, \tag{8a}$$

where

$$A(\mathbf{v},\mathbf{u}) = \int_\Omega [\mathbf{v}_x^T \mathbf{D}\mathbf{u}_x + \mathbf{v}_y^T \mathbf{D}\mathbf{u}_y + \mathbf{v}^T \mathbf{Q}\mathbf{u}] \, dxdy, \quad (\mathbf{v},\mathbf{u}) = \int_\Omega \mathbf{v}^T \mathbf{u} \, dxdy. \tag{8b,c}$$

As usual, the Sobolev space H^1 consists of functions having first partial derivatives in L^2. The subscripts E and 0 further restrict functions to satisfy the essential boundary conditions (7b) and trivial versions of (7b), respectively. Finite element solutions of (8) are constructed by approximating H^1 by a finite-dimensional subspace $S^{N,p}$ and determining $\mathbf{U} \in S_E^{N,p}$ such that

$$A(\mathbf{V},\mathbf{U}) + (\mathbf{V},\mathbf{f}) = \sum_{i=1}^{m} \int_{\partial\Omega_i^N} V_i c_i^N \, ds, \quad \text{for } \textit{all } \mathbf{V} \in S_0^{N,p}. \tag{9}$$

Selecting $S^{N,p}$ as a space of continuous piecewise p *th*-degree hierarchical polynomials [24] with respect to the partition of Ω into triangular finite elements, substituting these approximations into (9), and evaluating the integrals by quadrature rules yields a

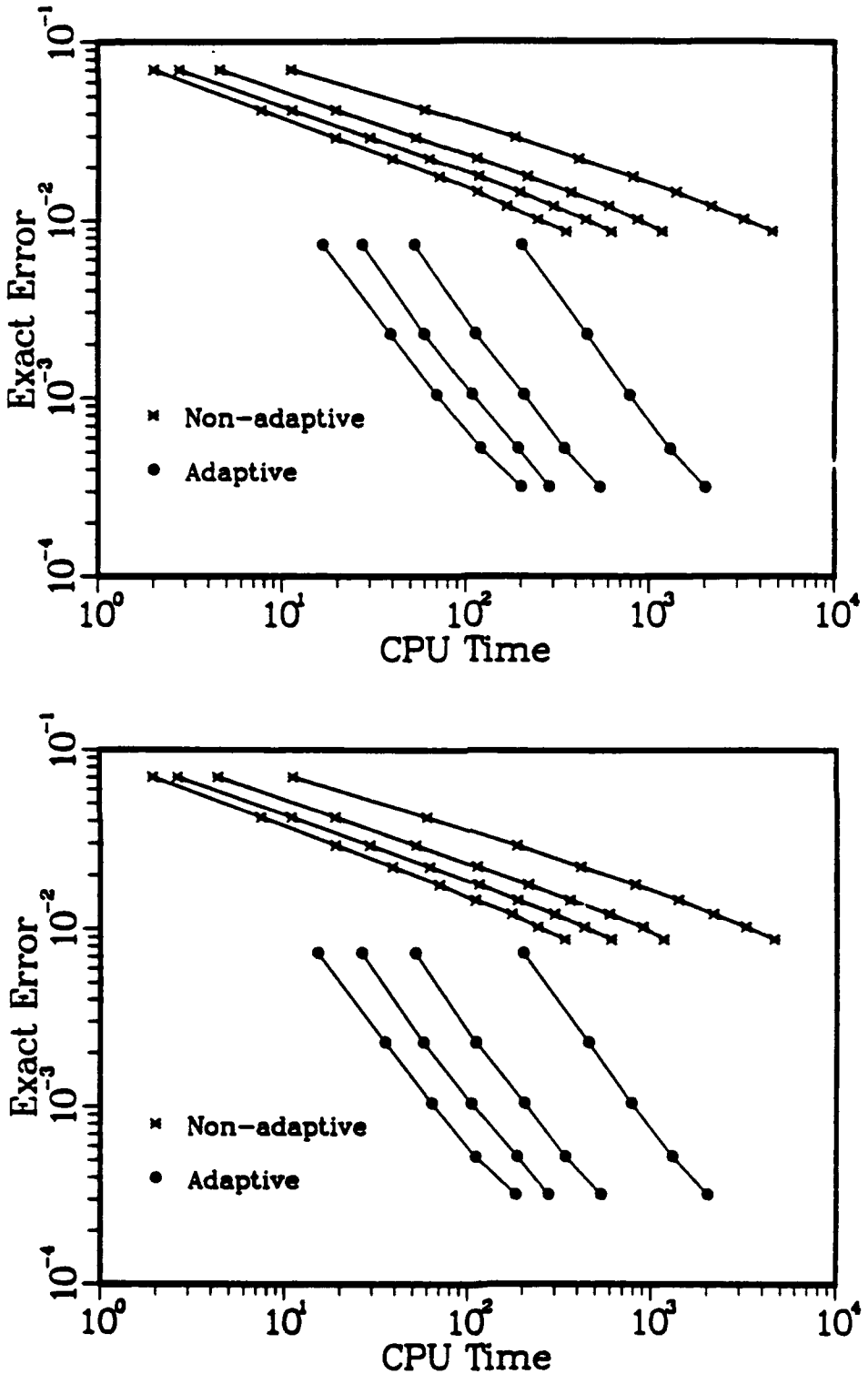

Figure 4. Global L^1 error as a function of CPU time for Example 1 using non-adaptive methods (upper set of curves) and adaptive h-refinement methods (lower set of curves) with static (top) and dynamic (bottom) load balancing. Computations are shown for systems using 1, 4, 8, and 15 processors (right to left for each set of curves).

sparse, symmetric, positive definite, N-dimensional linear system of the form

$$\mathbf{KX} = \mathbf{b}, \tag{10}$$

where $\mathbf{X}$ is an N-vector of Galerkin coordinates.

Meshes of triangular or quadrilateral elements are created automatically on Ω by using the *finite quadtree* procedure [11]. This structure is somewhat different than the tree of grids described in Section 2. With this technique, Ω is embedded in a square "universe" that may be recursively quartered to create a set of disjoint squares called *quadrants*. Data associated with these quadrants is managed by using a hierarchical tree structure with the original square universe regarded as the root and with smaller quadrants created by subdivision regarded as offspring of larger ones. Quadrants intersecting $\partial\Omega$ are recursively quartered until a prescribed spatial resolution of Ω has been obtained. At this stage, quadrants that are leaf nodes of the tree and intersect $\Omega \cup \partial\Omega$ are further divided into small sets of triangular or quadrilateral elements. Severe mesh gradation is avoided by imposing a maximal one-level difference between quadrants sharing a common edge. This implies a maximal two-level difference between quadrants sharing a common vertex. A final "smoothing" of the triangular or quadrilateral mesh improves element shapes and further reduces mesh gradation near $\partial\Omega$.

A simple example involving a domain consisting of a rectangle and a quarter circle, as shown in Figure 5, will illustrate the finite quadtree process. In the upper left portion of the figure, the square universe containing the problem domain is quartered creating the one-level tree structure shown at the upper right. Were this deemed to be a satisfactory geometrical resolution, a mesh of five triangles could be created. As shown, the triangular elements are associated with quadrants of the tree structure. In the lower portion of Figure 5, the quadrant containing the circular arc is quartered and the resulting quadrant that intersects the circular arc is quartered again to create the three-level tree shown in the lower right portion of the figure. A triangular mesh generated on this tree structure is also shown.

Arbitrarily complex two-dimensional domains may be discretized in this manner and generally produce unstructured grids; however, the underlying tree of quadrants remains regular. Adaptive mesh refinement is easily accomplished by subdividing appropriate leaf-node quadrants and generating a new mesh of triangular or quadrilateral elements locally; thus, unifying the mesh generation and adaptive solution phases of the problem under a common tree data structure.

Preconditioned conjugate gradient (PCG) iteration is an efficient means of solving the linear algebraic systems (10) that result from the finite element discretization of self-adjoint elliptic partial differential systems [8]. The key steps in the PCG procedure [22] involve (i) matrix-vector multiplication of the form

$$\mathbf{q} = \mathbf{Kp} \tag{11a}$$

and (ii) solving linear systems of the form

$$\overline{\mathbf{K}}\mathbf{d} = \mathbf{r}, \tag{11b}$$

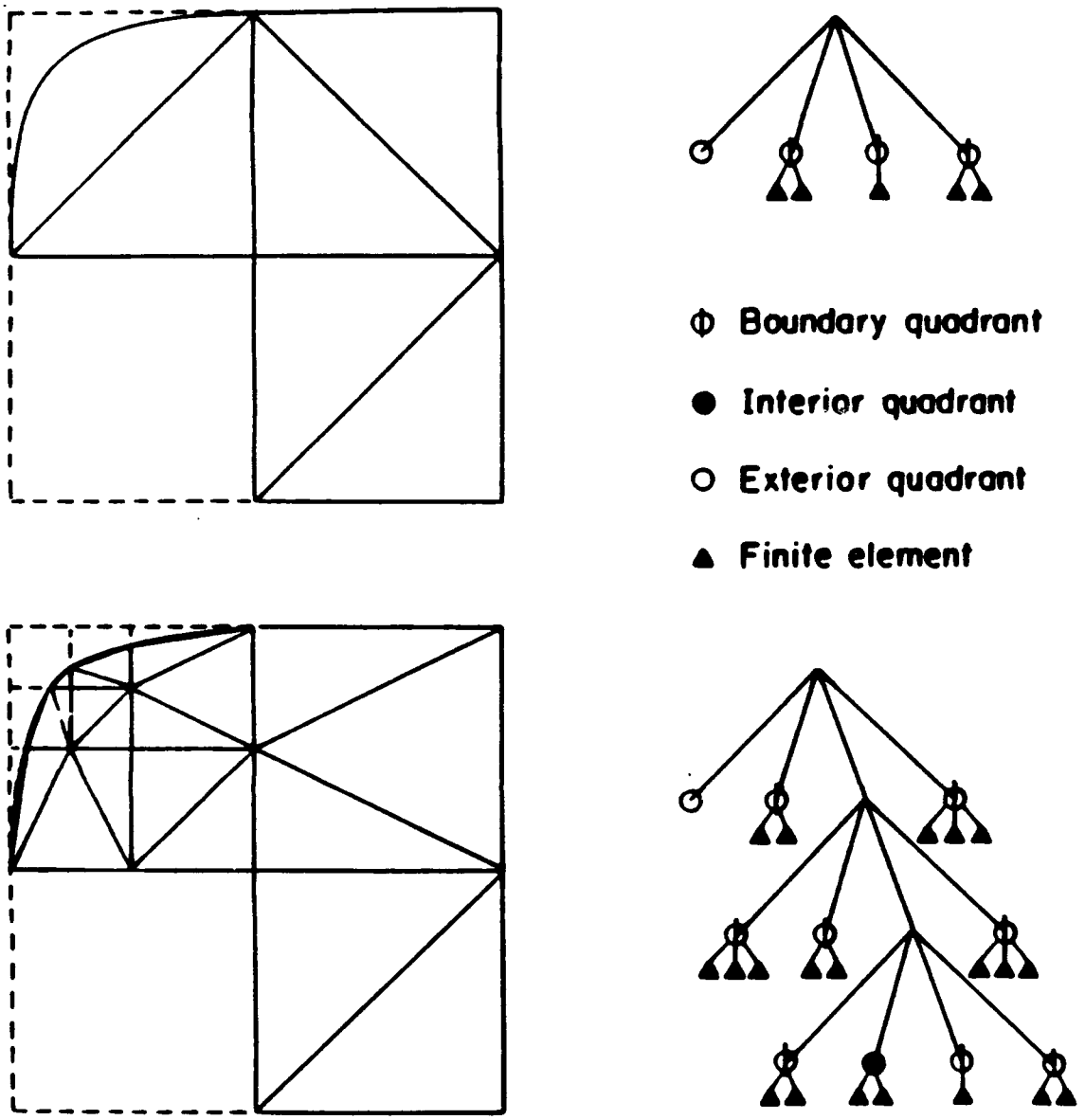

Figure 5. Finite quadtree mesh generation for a domain consisting of a rectangle and a quarter circle. One-level and three-level tree structures and their associated meshes of triangular elements are shown at the top and bottom of the figure, respectively.

where r and p are the residual vector and conjugate search direction, respectively. The preconditioning matrix $\overline{K}$ may be selected to reduce computational cost. The element-by-element (EBE) and symmetric successive over-relaxation (SSOR) preconditionings are in common use and seem appropriate for use with quadtree-structured grids. The EBE preconditioning is an approximate factorization of the stiffness matrix K into a product of elemental matrices. If the grid has been "colored" so as to

segregate non-contiguous elements, then (11b) can be solved in parallel on elements having the same color. Since the matrix-vector multiplication (11a) can also be performed in an element-by-element fashion, the entire PCG solution can be done in parallel on non-contiguous elements. While this simple approach has been used in several applications [18, 19, 21], we found the SSOR preconditioning to be more efficient in every instance [13] and, therefore, shall not discuss EBE preconditionings any further.

SOR and SSOR iteration have been used for the parallel solution of the five-point difference approximation of Poisson's equation on rectangular meshes by numbering the discrete equations and unknowns in "checkerboard" order [1]. With this ordering, unknowns at "red" mesh points are only coupled to those at "black" mesh points and vice versa; thus, solutions at all red points can proceed in parallel that may be followed by a similar solution at all black points. Preserving symmetry, as with the SSOR iteration, will make the SOR method a suitable preconditioning for the PCG method. Adams and Ortega [1] describe multicolor orderings on rectangular grids using several finite element and finite difference stencils. However, multicolor orderings for unstructured meshes are more difficult since nodal connectivity and difference stencils for high-degree polynomial approximations can be quite complex. The computational effort can be reduced when using quadtree-structured grids by considering multicolor orderings for block SSOR preconditionings at the quadrant level. To be specific, partition the stiffness matrix $\mathbf{K}$ by quadrants as

$$\mathbf{K} = \mathbf{D} - \mathbf{L} - \mathbf{L}^T \tag{12a}$$

where

$$\mathbf{D} = \begin{bmatrix} \mathbf{K}_{1,1} & & & \\ & \mathbf{K}_{2,2} & & \\ & & \cdots & \\ & & & \mathbf{K}_{Q,Q} \end{bmatrix}, \quad \mathbf{L} = -\begin{bmatrix} 0 & & & \\ \mathbf{K}_{2,1} & 0 & & \\ & & \cdots & \\ \mathbf{K}_{Q,1} & \mathbf{K}_{Q,2} & & 0 \end{bmatrix}. \tag{12b,c}$$

Nontrivial entries in a diagonal block $\mathbf{K}_{i,i}$ arise from Galerkin coordinates that are connected through the finite element basis to other unknowns in quadrant i. Nontrivial contributions to block $\mathbf{K}_{i,j}$ of the lower triangular matrix $\mathbf{L}$ arise when the support of the basis associated with a Galerkin coordinate in quadrant i intersects quadrant j.

Using an SSOR preconditioning, the solution of (11b) would be computed according to the two-step procedure

$$\mathbf{X}^{n+\frac{1}{2}} = \omega(\mathbf{L}\mathbf{X}^{n+\frac{1}{2}} + \mathbf{L}^T\mathbf{X}^n + \mathbf{r}) + (1 - \omega)\mathbf{X}^n, \tag{13a}$$

$$\mathbf{X}^{n+1} = \omega(\mathbf{L}^T\mathbf{X}^{n+1} + \mathbf{L}\mathbf{X}^{n+\frac{1}{2}} + \mathbf{r}) + (1 - \omega)\mathbf{X}^{n+\frac{1}{2}}, \quad n = 1, 2, \cdots, M. \tag{13b}$$

Thus, each block SSOR iteration consists of two block SOR steps; one having the reverse ordering of the other. Typically, $M = 3$ SSOR steps are performed between each PCG step.

Suppose that the Q quadrants of a finite quadtree structure are separated into γ disjoint sets. Then, using the symmetric γ-color block SSOR ordering, we would sweep the quadrants in the order $C_1, C_2, \cdots, C_\gamma, C_\gamma, C_{\gamma-1}, \cdots, C_1$, where C_i is the set of quadrants having color i. Because quadrants rather than nodes are colored, a node can be connected to other nodes having the same color. Thus, the forward and backward SOR sweeps may differ for a color C_i, $i = 1, 2, \cdots, \gamma$. During an SOR sweep, unknowns lying on quadrant boundaries are updated as many times as the number of quadrants containing them.

Coloring the regular quadrants of a finite quadtree is far simpler than coloring the elements of a mesh. Differences in the small number of elements within quadrants having the same color may cause some load imbalance and this effect will have to be investigated. Naturally, coloring procedures that use the fewest colors increase data granularity and reduce the need for process synchronization. At the same time, the cost of the coloring algorithm should not be the dominant computational cost. With these views in mind, we developed an eight-color procedure that has linear time complexity [13]. This procedure only required a simple breadth-first traversal of the quadtree, but performance never exceeded that of the six-color procedure which is described in the following paragraphs. Four-color procedures are undoubtedly possible, but we have not formulated any. Their complexity, unlike the eight- and six-color procedures, may be nonlinear.

With the aim of constructing a quadtree coloring procedure using a maximum of six colors, let us define a binary directed graph called a "quasi-binary tree" from the finite quadtree by using the following recursive assertive algorithm.

i. The root of the quadtree corresponds to the root of the quasi-binary tree.

ii. Every terminal quadrant is associated with a node in the quasi-binary tree; however, not every quasi-binary tree node must correspond to a quadrant.

iii. In the planar representation of the quadtree, nodes across a common horizontal edge are connected in the quasi-binary tree.

iv. When a quadrant is divided, its parent node in the quasi-binary tree becomes the root of a subtree.

Planar representations of simple quadtrees and their quasi-binary tree representations are illustrated in Figure 6. The leftmost quadtree illustrates root-node and offspring construction of the quasi-binary tree. Connection of nodes across horizontal edges is shown with and without quadrant division in all three illustrations. Subtree definitions according to assertion (iv) are shown in the center and rightmost quadtrees.

From Figure 6, we see that the column-order traversal of a finite quadtree is the depth-first traversal of its associated quasi-binary tree. Let us define six colors divided into three sets a, b, and c of two disjoint colors that alternate through the columns in a column-order traversal of the quadtree. Whenever left and right quasi-binary tree branches merge, column-order traversal continues using the color set associated with the left branch. Two of the three color streams, say a and b, are passed to a node of the quasi-binary tree. At each branching, the color stream a and the third color stream

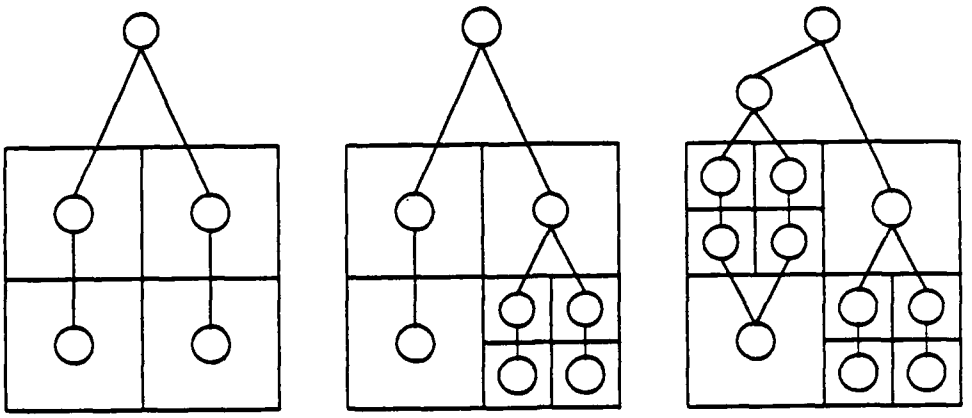

Figure 6. Planar representations of three quadtrees and their associated quasi-binary trees.

c are passed to the left offspring while the streams a and b are passed in reverse order to the right offspring. Additional details and a correctness proof of this algorithm will appear [15].

Computational experiments of Benantar et al. [13] demonstrate the excellent parallelism that may be obtained by the six-color SSOR PCG procedure with piecewise linear finite element approximations. However, higher-order polynomial bases create additional possibilities for processor load imbalance with coloring at the quadrant level. Let us illustrate this with a simple problem. As in Section 2, a 16-processor Sequent Balance 21000 computer was used for the experiment.

Example 2. Consider the Dirichlet problem

$$u_{xx} + u_{yy} = f(x,y), \quad (x,y) \in \Omega, \tag{14a}$$

$$u = 0, \quad (x,y) \in \partial\Omega, \tag{14b}$$

with $\Omega = \{ (x,y) \mid -3 < x, y < 3 \}$. We solved this problem on a 400-element mesh using piecewise linear, quadratic, and cubic approximations. Adaptive p-refinement with the polynomial degree p restricted to be 1, 2, or 3 was also performed. Parallel speed up and processor idle time resulting from the need to synchronize at the completion of each color are shown in Figure 7.

Parallel performance degrades as polynomial degree increases, with the adaptive strategy having the poorest performance. Adaptive algorithms typically have serial logic which limits speed up. Of course, speed up is not the only measure of complexity and an adaptive solution strategy could require less CPU time to solve the problem to a given level of accuracy. Nevertheless, additional research is necessary to improve

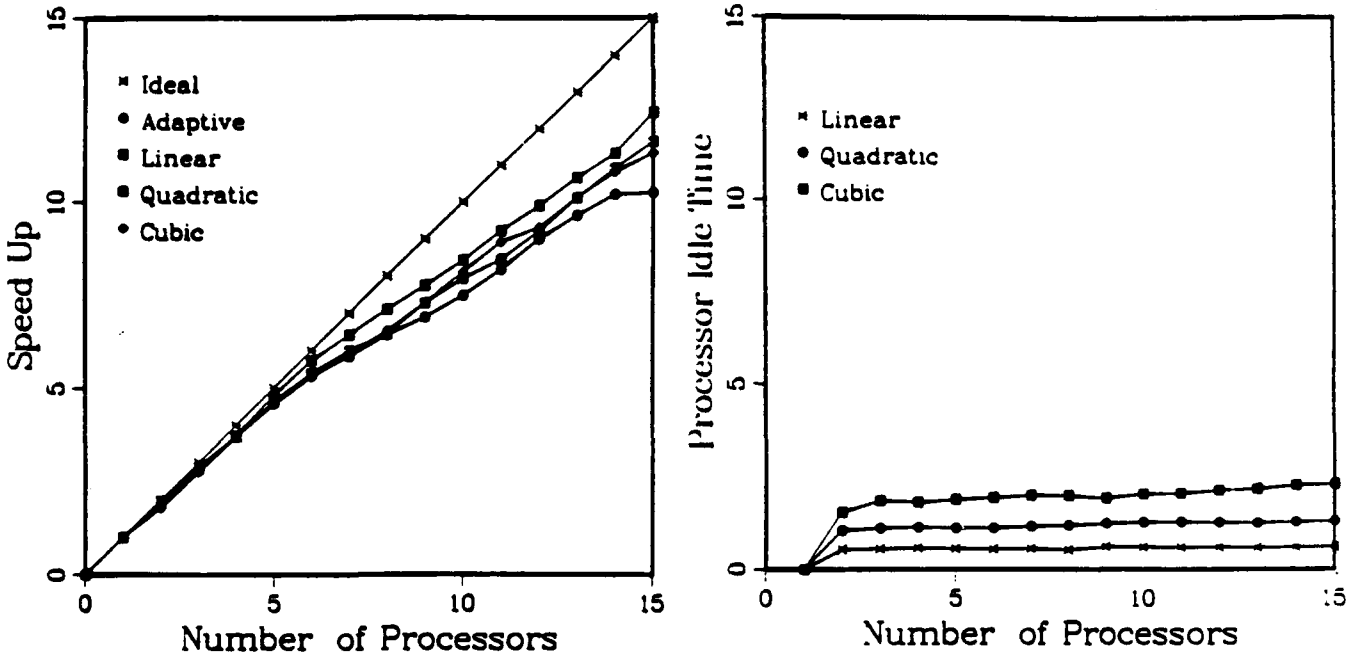

Figure 7. Parallel speed up (left) and processor idle time (right) for the finite element solution of Example 2 using piecewise linear, quadratic, and cubic approximations as well as adaptive p-refinement.

performance with high-order and adaptive strategies.

Using a hierarchical basis, all Galerkin coordinates for polynomial degrees higher than one are associated with mesh points that are either along element edges or within elements. Thus, the Galerkin coordinates for continuous piecewise linear approximations are the only ones associated with element vertices. Parallel performance could, therefore, be improved by coloring element edges rather than quadrants and we have designed a three-color procedure having linear time complexity to do this [15]. Since hierarchical bases add incremental corrections as the polynomial degree is increased, one could conceive an algorithm where quadrant coloring is used with the piecewise linear portion of the approximation and edge coloring is used for higher-degree approximations.

4. DISCUSSION. High-order and hp-refinement strategies have the highest convergence rates on serial processors. Successful use of adaptive strategies in parallel environments depends heavily on the efficient implementation of these procedures on shared- and distributed-memory computers. The edge coloring procedure alluded to in Section 3 should provide some improvement over existing strategies on shared-memory systems, but no procedure is available for using hp-refinement on data-parallel computers. High-order and hp-refinement techniques are being added to our collection of

methods for solving hyperbolic systems using the finite element methods of Cockburn and Shu [14]. The p-hierarchical Legendre polynomial basis embedded in these methods should also furnish error estimates similar to those that we have developed for parabolic systems [3]. These techniques are far more efficient than Richardson's extrapolation.

Our h-refinement procedure for hyperbolic systems could be improved by beginning each base-mesh time step with an adaptively chosen mesh that utilizes known nonuniformities in the solution discovered during the previous base-mesh time step. Processors would still have to be scheduled to balance loads in this case and procedures for doing this are unavailable. Finally, parallel procedures for distributed memory systems and procedures for three-dimensional problems are of great interest.

REFERENCES.

1. L. Adams and J. Ortega, A multi-color SOR method for parallel computation, in K. E. Batcher, W. C. Meilander, and J. L. Potter, Eds., *Proceedings of the International Conference on Parallel Processing*, Computer Society Press, Silver Spring, 1982, pp. 53-56.

2. S. Adjerid and J. E. Flaherty, A moving mesh finite element method with local refinement for parabolic partial differential equations, *Comput. Meths. Appl. Mech. Engr.* **56** (1986), pp. 3-26.

3. S. Adjerid, J. E. Flaherty, and Y. Wang, A posteriori error estimation with finite element methods of lines for one-dimensional parabolic systems, in preparation, 1990.

4. D. C. Arney, R. Biswas, and J. E. Flaherty, An adaptive mesh moving and refinement procedure for one-dimensional conservation laws, Tech. Rep. 90-6, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, 1990.

5. D. C. Arney and J. E. Flaherty, A two-dimensional mesh moving technique for time-dependent partial differential equations, *J. Comput. Phys.* **67** (1986), pp. 124-144.

6. D. C. Arney and J. E. Flaherty, An adaptive mesh-moving and local refinement method for time-dependent partial differential equations, *ACM Trans. Math. Softw.* **16** (1990), pp. 48-71.

7. U. M. Ascher, R. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, 1988.

8. O. Axelsson and V. A. Barker, *Finite Element Solution of Boundary Value Problems: Theory and Computation*, Academic Press, Orlando, 1984.

9. I. Babuska and E. Rank, An expert-system like approach in the hp-version of the finite element method, Tech. Note BN-1084, Institute for Physical Science and

Technology, University of Maryland, College Park, 1986.

10. I. Babuska, B. A. Szabo, and I. N. Katz, The p-version of the finite element method, *SIAM J. Numer. Anal.* **18** (1981), pp. 515-545.

11. P. L. Baehmann, S. L. Wittchen, M. S. Shephard, K. R. Grice, and M. A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, *Int. J. Numer. Meths. Engrg.* **24** (1987), pp. 1043-1078.

12. R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0*, Frontiers in Applied Mathematics 7, SIAM, Philadelphia, 1990.

13. M. Benantar, R. Biswas, J. E. Flaherty, and M. S. Shephard, Parallel computation with adaptive methods for elliptic and hyperbolic systems, *Comput. Meths. Appl. Mech. Engr.*, to appear, 1990.

14. B. Cockburn and C.-W. Shu, TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: general framework, *Maths. Comp.* **52** (1989), pp. 411-435.

15. J. E. Flaherty and M. Benantar, Parallel element-by-element techniques for elliptic systems using finite quadtree meshes, in preparation, 1990.

16. J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, *Adaptive Methods for Partial Differential Equations*, SIAM, Philadelphia, 1989.

17. C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, 1971.

18. I. Gustafsson and G. Lindskog, A preconditioning technique based on element matrix factorizations, *Comput. Meths. Appl. Mech. Engr.* **55** (1986), pp. 201-220.

19. R. B. King and V. Sonnad, Implementation of an element-by-element solution algorithm for the finite element method on a coarse-grained parallel computer, *Comput. Meths. Appl. Mech. Engr.* **65** (1987), pp. 47-59.

20. P. K. Moore and J. E. Flaherty, Adaptive local overlapping grid methods for parabolic systems in two space dimensions, Tech. Rep. 90-5, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, 1990.

21. B. Nour-Omid and B. N. Parlett, Element preconditioning using splitting techniques, *SIAM J. Sci. Stat. Comput.* **6** (1985), pp. 761-770.

22. J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

23. R. D. Richtmyer and K. W. Morton, *Difference Methods for Initial-Value Problems*, Interscience, New York, 1967.

24. B. Szabo and I. Babuska, *Introduction to Finite Element Analysis*, John Wiley and Sons, to appear, 1990.

TECHNICAL REPORT INTERNAL DISTRIBUTION LIST

	NO. OF COPIES
CHIEF, DEVELOPMENT ENGINEERING DIVISION	
ATTN: SMCAR-CCB-DA	1
-DC	1
-DI	1
-DR	1
-DS (SYSTEMS)	1
CHIEF, ENGINEERING SUPPORT DIVISION	
ATTN: SMCAR-CCB-S	1
-SD	1
-SE	1
CHIEF, RESEARCH DIVISION	
ATTN: SMCAR-CCB-R	2
-RA	1
-RE	1
-RM	1
-RP	1
-RT	1
TECHNICAL LIBRARY	5
ATTN: SMCAR-CCB-TL	
TECHNICAL PUBLICATIONS & EDITING SECTION	3
ATTN: SMCAR-CCB-TL	
OPERATIONS DIRECTORATE	1
ATTN: SMCWV-ODP-P	
DIRECTOR, PROCUREMENT DIRECTORATE	1
ATTN: SMCWV-PP	
DIRECTOR, PRODUCT ASSURANCE DIRECTORATE	1
ATTN: SMCWV-QA	

NOTE: PLEASE NOTIFY DIRECTOR, BENET LABORATORIES, ATTN: SMCAR-CCB-TL, OF ANY ADDRESS CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST

	NO. OF COPIES
ASST SEC OF THE ARMY RESEARCH AND DEVELOPMENT ATTN: DEPT FOR SCI AND TECH THE PENTAGON WASHINGTON, D.C. 20310-0103	1
ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER ATTN: DTIC-FDAC CAMERON STATION ALEXANDRIA, VA 22304-6145	12
COMMANDER US ARMY ARDEC	
ATTN: SMCAR-AEE	1
SMCAR-AES, BLDG. 321	1
SMCAR-AET-O, BLDG. 351N	1
SMCAR-CC	1
SMCAR-CCP-A	1
SMCAR-FSA	1
SMCAR-FSM-E	1
SMCAR-FSS-D, BLDG. 94	1
SMCAR-IMI-I (STINFO) BLDG. 59	2
PICATINNY ARSENAL, NJ 07806-5000	
DIRECTOR US ARMY BALLISTIC RESEARCH LABORATORY ATTN: SLCBR-DD-T, BLDG. 305 ABERDEEN PROVING GROUND, MD 21005-5066	1
DIRECTOR US ARMY MATERIEL SYSTEMS ANALYSIS ACTV ATTN: AMXSY-MP ABERDEEN PROVING GROUND, MD 21005-5071	1
COMMANDER HQ, AMCCOM ATTN: AMSMC-IMP-L ROCK ISLAND, IL 61299-6000	1
COMMANDER ROCK ISLAND ARSENAL ATTN: SMCRI-ENM ROCK ISLAND, IL 61299-5000	1
DIRECTOR US ARMY INDUSTRIAL BASE ENGR ACTV ATTN: AMXIB-P ROCK ISLAND, IL 61299-7260	1
COMMANDER US ARMY TANK-AUTMV R&D COMMAND ATTN: AMSTA-DDL (TECH LIB) WARREN, MI 48397-5000	1
COMMANDER US MILITARY ACADEMY ATTN: DEPARTMENT OF MECHANICS WEST POINT, NY 10996-1792	1
US ARMY MISSILE COMMAND REDSTONE SCIENTIFIC INFO CTR ATTN: DOCUMENTS SECT, BLDG. 4484 REDSTONE ARSENAL, AL 35898-5241	2
COMMANDER US ARMY FGN SCIENCE AND TECH CTR ATTN: DRXST-SD 220 7TH STREET, N.E. CHARLOTTESVILLE, VA 22901	1
COMMANDER US ARMY LABCOM MATERIALS TECHNOLOGY LAB ATTN: SLCMT-IML (TECH LIB) WATERTOWN, MA 02172-0001	2

NOTE: PLEASE NOTIFY COMMANDER, ARMAMENT RESEARCH, DEVELOPMENT, AND ENGINEERING CENTER, US ARMY AMCCOM, ATTN: BENET LABORATORIES, SMCAR-CCB-TL, WATERVLIET, NY 12189-4050, OF ANY ADDRESS CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST (CONT'D)

	NO. OF COPIES
COMMANDER US ARMY LABCOM, ISA ATTN: SLCIS-IM-TL 2800 POWDER MILL ROAD ADELPHI, MD 20783-1145	1
COMMANDER US ARMY RESEARCH OFFICE ATTN: CHIEF, IPO P.O. BOX 12211 RESEARCH TRIANGLE PARK, NC 27709-2211	1
DIRECTOR US NAVAL RESEARCH LAB ATTN: MATERIALS SCI & TECH DIVISION CODE 26-27 (DOC LIB) WASHINGTON, D.C. 20375	1 1
DIRECTOR US ARMY BALLISTIC RESEARCH LABORATORY ATTN: SLCBR-IB-M (DR. BRUCE BURNS) ABERDEEN PROVING GROUND, MD 21005-5066	1
COMMANDER AIR FORCE ARMAMENT LABORATORY ATTN: AFATL/MN EGLIN AFB, FL 32542-5434	1
COMMANDER AIR FORCE ARMAMENT LABORATORY ATTN: AFATL/MNF EGLIN AFB, FL 32542-5434	1
MIAC/CINDAS PURDUE UNIVERSITY 2595 YEAGER ROAD WEST LAFAYETTE, IN 47905	1

NOTE: PLEASE NOTIFY COMMANDER, ARMAMENT RESEARCH, DEVELOPMENT, AND ENGINEERING CENTER, US ARMY AMCCOM, ATTN: BENET LABORATORIES, SMCAR-CCB-TL, WATERVLIET, NY 12189-4050, OF ANY ADDRESS CHANGES.